THE WHITE-HAT BOT: A NOVEL BOTNET DEFENSE STRATEGY

THESIS

Tyrone C. Gubler, Technical Sergeant, USAF

AFIT/GCS/ENG/12-05

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AIFT/GCS/ENG/12-05

THE WHITE-HAT BOT: A NOVEL BOTNET DEFENSE STRATEGY

THESIS

Presented to the Faculty

Department of Electrical and Computer Engineering

Graduate School of Engineering and Management

Air Force Institute of Technology

Air University

Air Education and Training Command

In Partial Fulfillment of the Requirements for the

Degree of Master of Science

Tyrone C. Gubler, BS

Technical Sergeant, USAF

June 2012

AIFT/GCS/ENG/12-05

THE WHITE-HAT BOT: A NOVEL BOTNET DEFENSE STRATEGY

Tyrone C. Gubler, BS

Technical Sergeant, USAF

Approved:

[illegible]
Lt Col Jeffrey M. Hemmes, Ph.D., USAF (Chairman)

5 Jun 2012
Date

[illegible]
Maj Jonathan W. Butts, Ph.D., USAF (Member)

29 May 2012
Date

Douglas D. Hodson
Dr. Douglas D. Hodson (Member)

5 JUN 2012
Date

AIFT/GCS/ENG/12-05

Abstract

Botnets are a threat to computer systems and users around the world. Botmasters can range from annoying spam email propagators to nefarious criminals. These criminals attempt to take down networks or web servers through distributed denial-of-service attacks, to steal corporate secrets, or to launder money from individuals or corporations. As the number and severity of successful botnet attacks rise, computer security experts need to develop better early-detection and removal techniques to protect computer networks and individual computer users from these very real threats. I will define botnets and describe some of their common purposes and current uses. Next, I will reveal some of the techniques currently used by software security professionals to combat this problem. Finally, I provide a novel defensive strategy, the White-hat Bot (WHB), with documented experiments and results that may prove useful in the defense against botnets now and in the future.

Acknowledgments

First and foremost, I would like to thank my Heavenly Father, without whom none of this would be possible. Next, I would be remiss in neglecting to express my gratitude to my wife, for her unwavering support throughout my degree program. Thanks for standing by me; I know it was not easy dealing with the long days and nights of neglect while I studied. Also, I would like to express my sincere appreciation to my faculty advisor, Lt Col Jeffrey Hemmes, for his guidance and support throughout the course of this thesis effort. Additionally, I would like to thank Raytheon's BBN Technologies division for sharing their SLINGbot programming framework with me. And finally, thank you Mitre Corporation for sharing your Enhanced SLINGbot framework. These software frameworks have proven to be a valuable part of my research effort.

Tyrone C. Gubler

Table of Contents

List of Figures

List of Tables

THE WHITE-HAT BOT: A NOVEL BOTNET DEFENSE STRATEGY

I. Introduction

General Issue

The goal of this research is to explore the problem domain of botnets and to present a novel approach, the White-hat Bot (WHB), for their detection. This research focuses on the distributed nature of botnets and explores how botnet technology can be used to locate bot client and bot server applications by analyzing the incoming and outgoing network messages on the bot client or server machines.

Some estimates show that as many as 40 percent of all computers are part of a bot network (Zhu et al., 2008). Anti-virus software can detect known bot software, firewall software can block specified ports and applications, and intrusion detection systems (IDS) can detect and block known exploits. When a new botnet exploit is created, many times it will go undetected and have a wide range of negative effects for a period of time before being reported, analyzed, and a subsequent remedy created and deployed on a large scale to prevent future infection. It is this delay that I attempt to minimize with the WHB by providing a way to more quickly detect novel threats.

The concept primarily consists of analysis of the network traffic on any machines that are suspected of an unknown infection. To determine if a machine is suspect is a difficult problem that requires input from the user's machine, but that input can be unspecific such as the machine running slower than usual, or an application running in the background that was not authorized, either directly or indirectly, by any legitimate user or administrator of that machine. Undetermined anomalous behavior can be detected by automated sensors on the user's machine, such as by some anti-virus software, firewall software, IDS software, system memory analysis tools, disk usage monitoring tools, etc. (Chandrashekar, Orrin, Livadas, & Schooler, 2009).

Certain botnet exploits disable anti-virus software and prevent that software from downloading and installing the latest virus definitions, leaving them vulnerable to some known threats (Chandrashekar et al., 2009). One potential problem indicator could be determined by a tracking mechanism implemented at the anti-virus vendor's location (or at a local administrator's level for a network of nodes) to determine the subset of machines that have not, for whatever reason, downloaded the latest anti-virus signatures or other security patches and updates. These nodes may automatically be considered suspect if there is no valid reason, such as the machine being powered down or disconnected from the network, for the lack of timely updates.

This tracking mechanism could be as simple as a check-in feature that identifies each node by a serial number assigned to that node's copy of the anti-malware software whenever the application connects to the update server to download the latest patch, signatures, or updates. This would allow the vendor to maintain a database that could be used to identify users with potential problems. For the purposes of this research it is assumed that there is some sort of reporting mechanism in place, and that the experimental nodes are all considered suspect. This allows the focus of the research to be on the analysis of network traffic for the detection of the specific novel applications that are responsible for the undesired botnet behavior.

How real is the threat of botnets? Norman Elton and Matt Keel called bot networks "the single greatest threat facing humanity" (Schiller et al., 2007). While there are no doubt greater ills in the world such as the deterioration of the family, poverty and violent crime, botnets are nonetheless real problems that must be taken seriously.

One paper focuses on the threat posed by Al Qaeda operatives who possess interest in cyber terror plots (The Lipman Reports Editors, 2010). These operatives currently use websites and chat rooms to recruit, teach terrorist skills such as bomb building, and other nefarious activities. The authors suspect that these terrorists may be behind some of the recent computer network attacks in South Korea and elsewhere. The authors further speculate that these could be "dry runs" in preparation for a massive attack that may include physical attacks combined with cyber attacks. A recent government security exercise involved an attack via 20 million compromised smart phones. With the current widespread popularity of such devices, this type of scenario seems very plausible.

The number of existing botnets is indeterminate because there are potentially untold millions of bots, both active and inactive, that have yet to be discovered. Numerous new machines are infected and added to the list on a daily basis. "MessageLabs Intelligence tracks at least 21,000 active spam-sending bots in the US in an average week" (Wood, 2010). Between "January and June 2006, Symantec…observed more than 4.5 million distinct, active bot network computers" (Schiller et al., 2007). "The anti-botnet vendor FireEye…" posts even higher estimates of "…150 million bot-infected computers worldwide" (Hunter, 2008).

There are many exploits perpetuated by botnets such as targeted distributed denial of service (DDoS) attacks, malware dissemination, spam message generation, identity theft, etc. I will briefly discuss one common exploit here. One fraudulent activity perpetuated by botnets involves ad services such as Google's AdSense which pays website owners revenue for posting the AdSense banner on their web site (Google, 2012).

The AdSense banner displays messages from legitimate business websites, and Google pays the site owner a per-click fee for every legitimate visitor who clicks an advertisement banner. The botmaster creates a bot that is programmed to visit the botmaster's own websites to click on the advertisements displayed in the AdSense banners. Since these clicks originate from IP addresses around the world, i.e., the botmaster's infected clients, they are seen as legitimate traffic and the botmaster is paid for each click. This technique is commonly known as click fraud. "Once criminals start to build their click fraud network, they can start earning real money" (Ollmann, 2009). There are many other exploits possible, but the focus of this research is on the detection of botnets, so the discussion will be to that end.

The objective of this research is to determine the value of botnet technology retooled toward the detection of malicious botnets. The hypothesis is that the distributed nature of botnets provides a useful advantage over current methods of botnet detection. One of the primary advantages of this methodology can be leveraged by the fact that botnets are not limited to a specific administrative domain. A botnet has the ability to span an arbitrary number of domains, as does the WHB, and as such the WHB has the potential to be used on every machine within one or more malicious botnet domains. Under the right circumstances this provides the WHB the ability to operate on every infected node of a malicious botnet, to include all of that botnet's Command and Control (CC) servers. This provides the WHB the potential to completely eradicate a malicious botnet. If the WHB is only installed on a subset of a given botnet, then the data collected from that subset can be shared with other anti-malware companies in order to detect the botnet software on other non-WHB nodes.

Investigative Questions

Will the WHB be more effective than current methods, such as typical bot-detection software, to locate zero-day malicious bot network applications running on infected hosts? If so, what methods and best practices must be applied to make this possible?

Will the distributed nature of botnets and the methods employed by malicious bot masters be beneficial in the detection of novel bot threats?

Can the benefits of current tools be leveraged and amplified by their use in a distributed bot network? If so, how?

If these methods do prove to be more effective, they will be a valuable tool for the Department of Defense and for the entire malware defense industry.

Summary

The threat botnets pose to computing systems of all types is undeniable and imminent. The relevance of this threat was summed up by President Barack Obama when he said, “So cyberspace is real. And so are the risks that come with it. It’s the great irony of our Information Age—the very technologies that empower us to create and to build also empower those who would disrupt and destroy. And this paradox—seen and unseen—is something that we experience every day” (2009).

This research effort attempts to minimize the threats botnets pose by implementing a novel botnet detection strategy, the White-hat Bot, toward the detection of novel botnet threats. The WHB leverages the distributive nature of botnets, combined with current tools, to successfully detect the presence of malicious botnet software.

II. Literature Review

Chapter Overview

This chapter presents an overview of the botnet problem and explores a representative cross-section of the existing body of research in the area of botnet defense strategies, with a focus on current methods of botnet detection. This is done in preparation for the experimentation and botnet research study that follow.

A Botnet Survey

My definition of a botnet: a botnet is an extension of the concept of malicious software, such as a virus or worm, in which the malware provides a CC communication channel between the malicious code and the botmaster or bot-herder who uses this communication channel to control the botnet. According to malware data analyst Dan Bleaken of Symantec Hosted Services, "Botnets are distributed networks of 'zombie' or 'bot' PCs, infected by malware which enables them to be marshaled by cybercriminals – primarily to distribute enormous volumes of spam and other malware and launch phishing attacks via email" (Bleaken, 2010). The ability of this specialized malware to communicate creates opportunities for the attacker and it provides novel threats to the computing community at large.

Botnets are often classified by their CC architecture (Feily, Shahrestani, & Ramadass, 2009). The common architectures are IRC (Internet Relay Chat), HTTP (Hypertext Transport Protocol), DNS (Domain Name System), and P2P (Peer to Peer) based architectures.

With IRC botnets, the bots connect to a predetermined chat channel on an IRC server that the CC server also joins. The CC server sends commands to this channel, and the bots execute these commands and post their responses back to the same channel. This is the most popular method of communication for botnets, primarily because of the number of open source IRC applications available and the simplicity of deploying one. There are also numerous free web hosting services that allow anonymous vectors to set up chat channels, many of which are not actively monitored for botnet collaboration.

HTTP botnets communicate through standard web servers. HTTP botnets are the latest variety to become popular with cyber criminals, with the first web based botnets architectures appearing in 2005 (Koo, Chang, & Wei, 2011). The first of the HTTP based BlackEnergy botnet variants was located the following year. Free web server libraries and frameworks make creation of simple to complex web servers easy and cheap or even free in many cases. With the abundance of free web hosting available around the globe, it is trivial to deploy a free web server without necessarily knowing in what state, or even in what country, the physical hardware is located. HTTP bots may use a centralized web server or a distributed hierarchical set of web servers in order to communicate with their botmaster. Messages are formed as HTTP packets and often use standard internet traffic ports, but they are still free to use any port of their choice. Using standard internet ports may allow the application to avoid detection, as any traffic on these ports is typically trusted by default by firewall and IDS tools (Singh, Srivastava, Giffin, & Lee, 2008).

DNS based botnets use proxy servers in an effort to hide the true location of the CC server (Zhu et al., 2008). A proxy server acts as a communication intermediary by forwarding messages while appearing to either side as the originator or sink for each message. In this way, the IP address of the CC server remains hidden, and a particular bot appears only to communicate with the proxy server. Fast-flux is an application that is often used for this purpose (Zhu et al. 2008). It allows for a list of compromised computer's IP addresses to periodically rotate for a particular DNS record, allowing one DNS record to alternate proxy servers over time. This increases the difficulty of tracing the traffic path from a bot to the controlling server.

The P2P botnet architecture propagates commands through the botnet using some structured or unstructured algorithm. This architecture can vary greatly and may emulate the message passing structures seen in certain popular P2P applications such as Napster or Gnutella, and others (Grizzard, Sharma, Nunnery, Kang, & Dagon, 2007). In a hierarchical P2P configuration, the botmaster may send and receive all messages through one or more superpeers. That superpeer then forwards the message to a small group of peers at a lower echelon, who in turn propagate them to an arbitrary number of lower level peers until the message has eventually reached all bots. Another important part of this type of network is a mechanism to allow new peers to join and to facilitate peers who fail or otherwise leave the network. Redundant network overlays and alternate peer lists may help to keep the network stable in the face of uncertainty.

None of the communications frameworks mentioned here are mutually exclusive. It is possible to build hybrid botnets that incorporate two or more of these communications methodologies in complex ways in order to create a robust network. For instance, a P2P botnet may incorporate an HTTP communications construct, and use Fast-flux to rotate the IP addresses of the DNS records that point to its CC servers.

Botnet Design Notes

As anti-bot technology has advanced, so have evasion techniques. Regardless, anti-virus and anti-spyware technologies have come a long way in recent times. Typically, anti-virus programs detect the signatures of known malware and subsequently quarantine and eradicate the program on the infected machine. Bot creators are fully aware of this technique; hence, many of the latest bots are modular and polymorphic in design. Modularity provides the bot with the ability to be sent as a very small initial infection program. Once the initial program has infected a machine, it communicates this achievement to the botmaster. The botmaster can then forward the next software module to the bot through a communication channel of choice. This secondary module often incorporates new capability that is often used to disable any running anti-virus, firewall, and anti-spyware programs, making the victim machine a softer target (Chandrashekar et al., 2009). Some bots institute website blacklists to disallow the user from going to anti-virus websites, or any other site that may be considered detrimental to the life of the bot. Once this stage succeeds, the botmaster is again notified to send the next module (Schiller et al., 2007).

In the next phase, another module may be sent and installed to further enhance capabilities of the bot. Now that the system's defenses are down, the serious payloads can be more easily executed and are less likely to be compromised. The botmaster can install spambot functionality, key and click logging programs, some of which are capable of overcoming the on-screen keyboards used by some banks and government pay system websites by returning a screen capture for each mouse click, among myriad other capabilities. TeamDev's Java API, JxCapture, is an example of a software library that provides this type of functionality, for both still images and live video screen capture functions (TeamDev, 2012). Also, self-destruction and stealth functionality can be achieved. File systems can be scanned and sensitive files can be sent back to the botmaster with relative ease. Bot modularity provides the botmaster the ability to add new functionality, at will, to any or all of her bots as the need arises.

Polymorphic design allows the bot to evolve over time, altering its own signatures just enough to evade detection by the applications that might detect and eradicate it. This advent makes it even more difficult for security software developers to combat.

From this point on, I will refer to anti-virus vendors, anti-spyware vendors, firewall providers, etc., as security software providers (SSP), unless individual identification is necessary to stress a particular point.

An interesting variant of the typical botnet is called the “roving bugnet”. This botnet focuses on a single, high-value target: specifically an individual person. The concept behind the bugnet is that users can be assumed to have multiple network-connected devices with cameras and microphones. If a would-be attacker were to install a bot on an individual’s cell phone, laptop, desktop, and work PC, it would be plausible for them to take over these devices’ cameras and microphones to monitor just about everything the victim does. Ryan Farley and Xinyuan Wang created such a bugnet that was fully functional. As long as the target was “in the physical proximity of a compromised device for a microphone to pick up the victim’s sound,” the botmaster was able to monitor the activities of that victim (Farley & Wang, 2010).

Farley and Wang were able to deploy the bot on systems from Windows 95 to Vista, and Mac OS X. Additionally, they were capable of dynamically infecting and taking over additional devices belonging to the user. Farley and Wang's research illustrates the flexibility of botnets and why they are an attractive adversarial tool sought after by many groups.

Relevant Research

BotHunter

BotHunter is a freely available bot detection tool, created by a U.S. Army research effort, which advertises the ability to detect and to profile novel bot networks by network packet analysis (Gu, Porras, Yegneswaran, Fong, & Lee, 2007). This tool uses a "heavily modified and customized" version of the Snort IDS, combined with a correlation engine used to "model dialog production patterns against an abstract malware infection lifecycle model" (SRI International, 2012). BotHunter is used to detect novel botnets without the use of known signatures and without any a priori knowledge of the specific botnet modeled.

Sevy retooled and deployed Cybercraft, a Department of Defense project, as a botnet and passed botnet command messages within the normally unused media access control (MAC) address field of address resolution protocol (ARP) messages (Sevy, 2009). By applying this methodology, he was able to evade detection by BotHunter. This work shows that tools such as BotHunter are not always able to detect novel botnets and that more research is necessary to combat this growing problem.

BotHunter is designed to operate on a single edge node of a given network. This edge node is used to monitor all incoming and outgoing network messages. Using this methodology does not produce the distributed gains attained by the WHB implementation. Another weakness of this methodology is that the monitor node becomes a single point of failure and creates a bottleneck in the network. The performance of the network, and the performance of the detection operations, may suffer during peak traffic times if packets are dropped or ignored for the lack of available resources on the monitor node due to an overload condition.

In this research effort, BotHunter is used in an attempt to detect the Enhanced SLINGbot (System for Live Investigation of Next Generation Botnets) simulated malicious botnets. This trial will be used as a baseline comparison in order to determine the effectiveness of the WHB.

Anti-virus Applications

Typical anti-virus software performs static analysis of files on a computer. These applications look for specific signatures of known malicious binaries. These methods have proven useful in determining the presence of known threats. Unfortunately, they do not address new threats in a timely manner. New threats must be reported, the binaries must be captured, and an anti-virus signature created and published to all users before the threat is contained.

Kaspersky Anti-Virus is one of the many commercial anti-virus products available on the market today. It advertises both a static and dynamic analysis capability. The static analysis evaluates the binaries and detects specific signatures within those binaries, with a high rate of precision, only for known threats. Additionally, they apply a heuristic method within their static analysis to attempt the detection of novel threats. This analysis counts the number of potentially suspicious system calls within the binary. If the number of these suspicious calls exceeds some preset threshold, the application is then marked as suspicious and the user is notified. However, the disclaimer for this heuristic methodology is that “the detection rate for new malicious code is low, while the false positive rate is high” (Kaspersky Lab, 2012).

Kaspersky’s dynamic analysis mechanism emulates execution of a portion of the application within a sandboxed environment prior to allowing the executable to be executed as normal by the system. During this sandboxed execution mode, detection of anomalous behavior is attempted and logged. When this method is combined with the static heuristic scans, they claim that the detection rates are higher and the false positive rates are much lower than with static scans alone for novel threats. The drawback to this analysis is that the user must wait for this scan to take place before the system allows their application to execute. Additionally, “the dynamic method requires significantly more system resources than the static method” (Kaspersky Lab, 2012).

While the WHB does require the use of some system resources on the user’s machine, it does not require any wait for the user to start any applications on their PC. Further, the WHB focuses on the way in which applications communicate with processes on other machines in order to detect malicious use. Hence, Kaspersky and other SSP applications are still necessary to detect malware that does not communicate, such as worms and viruses, but for malware that communicates, other solutions, such as the WHB may make better sense since they focus upon these communications to detect the botnet software present.

While anti-virus software is likely to always have a place in malware detection for known threats, and even some novel threats, other detection methods will be useful to detect threats such as novel botnets. One way in which botnets can evade detection from anti-virus software is by dynamic recompilation using a random code generator in order to change the signatures of their binaries (Muhaya, Khan, & Xiang, 2011). The WHB addresses these concerns with the ability to detect botnets based upon the way they communicate with other nodes, regardless of the state of their compiled binaries. The hypothesis is that even in the face of a code generator changing a particular bot's binaries, the botnet's network messages must still conform to a specific interface format in order for the network to continue to communicate effectively. It is these communications packets that will not change with a recompilation of a given bot, without a synchronized recompilation of the entire bot network, which is not impossible, but a highly unlikely event. When an entire botnet does change the layout of their messages within the system, we can still rely on the fact that the WHB may still be able to detect these messages since the detection rules may be very generalized to find patterns in the messages that will likely still exist.

Firewall Software

Firewall software blocks messages from specific communications ports and specific applications from entering a network or a specific machine. Like anti-virus scanning, this methodology can only protect against known threats, and only to a certain extent. To get around a firewall, an application may simply find a non-standard port that is not blocked by the firewall application and use a different application name or IP address to send its messages. Alerts from a firewall application may be able to warn that a specific IP address is sending a message to a specific port, and it may subsequently block that port, but the application may then simply try another port.

Using the standard version of Snort, the WHB can scan all network messages, regardless of the port used or the application responsible for sending the message. This provides the WHB the ability to detect malicious messages, even if the sending application dynamically alters the communications port used from time to time.

Honeypots

Honeypots or honeynets are computers or networks that are set up by researchers or security experts to attract botnets and other forms of malware. Often these honeypots are implemented with sandboxed virtual machines to help reduce the chances that the honeypot itself can be used by perpetrators to launch actual attacks on innocent victims. They sometimes employ a black-hole approach to most outgoing communications to avoid the damage that may be caused to others by the infected honeypot's outgoing messages (Feily et al. 2009). In this way, a bot installed on a honeypot node can receive instruction from its CC, but messages from the bot will not be delivered. The OS will not report this failure to send to the bot application; thus, as far as the bot application knows, its outgoing messages were sent by the OS. One example of an ongoing honeypot effort is the Honeynet Project, "…a leading international security research organization, dedicated to investigating the latest attacks and developing open source security tools to improve Internet security" (Honeynet Project, 2012).

Honeypots lure in malware by purposefully leaving wide gaps in the security of the installation of the system. Some of the methods used to accomplish this are by installing outdated operating systems without the latest updates and patches, and by neglecting to install any SSP applications. The researchers then monitor the node or network in order to discover the exploits that are attempted and to capture the malware that is installed. This is one of the primary methods used by SSPs to capture malware in the wild in order to create new detection signatures and tools.

One problem with honeypots is that many of the more advanced malware applications have devised ways to detect honeypots in an effort to avoid them. In fact, many malware variants include the capability of detecting virtual machine environments, and when detected, the application opts out of infecting that node because these environments are typically not useful to the botmaster. Honeypot avoidance is critical to the longevity of the malware, since installation on a honeypot will ultimately result in the publication of defensive measures to defeat it.

The WHB addresses honeypot avoidance measures by operating on the actual systems of live users, or on the data servers that users frequently access. In contrast to a honeypot, the assumption is that the typical user will apply at least some set of standard defensive measures, such as SSP applications. Since some defensive measures are likely, any malware installed on their machine is more likely to either be novel and thus undetectable with many of the current standard methods, or advanced enough to evade or disable typical system defensive measures.

Regardless of their potential weaknesses, honeypots have proven useful in the field of novel botnet detection. Even though the WHB is geared for use on live networks, it too is a tool that may be used to monitor honeypot nodes in order to aid in the discovery of novel botnets.

Summary

This chapter presented an overview of the botnet problem and some of the tactics used to propagate malware. I discussed some of the tools and techniques currently used to defend against this threat, including BotHunter, KasperSky anti-virus, and honeypots.

Now that a wide swath of information regarding the botnet phenomenon has been covered, I will move on to a discussion of the WHB, the tools used in and rationale behind its construction, the experimental setup, and how these results will be measured.

III. Methodology

Chapter Overview

The purpose of this chapter is to review the tools and techniques used in building and evaluating the effectiveness of the WHB. First, I discuss the Snort Intrusion Detection System (Snort IDS) which captures and conducts analysis of network communications packets to detect potential malicious content. Then, I will discuss the SLINGbot (System for Live Investigation of Next Generation Botnets) and Enhanced SLINGbot code frameworks generated by prior research in the field by Raytheon and Mitre corporations, respectively. Next, I discuss the WHB designed as a part of this research from the Enhanced SLINGbot code base with integration of the Snort IDS system. Then, I discuss the Emulab testbed which will be used for large scale tests of the WHB concept. Next, I lay out the experimental setup and evaluation plans. Lastly, I cover the plans to compare the WHB concept to other existing botnet detection methods and tools.

Snort

Snort, is an intrusion detection system (IDS) that detects malicious activity by analyzing network packets and comparing them against a set of rules (Wuu & Chen, 2003). Because all botnet software communicates through network protocol messages, Snort is a logical fit for their detection. Snort is used by BotHunter (SRI International, 2012), and it has been used in other bot detection research efforts (Wuu & Chen, 2003; Still, 2011). Snort software is capable of operating on a local network to analyze every packet on the network (inline configuration), or it can be configured to operate on an individual node, only analyzing packets sent to and from that node (passive mode). In the WHB configuration, the tool operates on individual nodes in passive mode, scanning only the traffic unique to that node, be it the origin or terminus node for a given communications packet. In this manner, each host in a distributed bot network can report on its individual status to the CC node. This reporting provides for micro analysis of each host at the node level and for macro analysis of the entire network, by aggregation of the micro data, at the bot network level. While Snort was the chosen tool used with the WHB during this research effort, the WHB can easily be reconfigured to use any arbitrary payload and to utilize any number tools to accomplish its mission.

One goal is consolidation of the results of packet capture from multiple nodes in an effort to find common communication end points. When this is done, it should, in theory, show that multiple hosts frequently communicate with a subset of specific IP addresses. Granted, this will happen naturally between hosts and legitimate and popular web servers, however, illegitimate bot servers may be identified more easily by ignoring servers known to be legitimate. Although, caution should be exercised when ignoring known servers that provide legitimate services, since these legitimate services may also be exploited and used by illegitimate processes or used by persons for illegitimate purposes. For instance, the server for a popular email service, such as Gmail, may be exploited to carry CC messages for a botnet (Singh et al., 2008). Hence most of the messages to and from a server may be for legitimate uses, but some subset of the messages may contain malicious content related to a botnet. For this reason, a rule should not be employed to completely ignore all such messages from known legitimate sources without risking the possibility of missing botnet CC messages. A better policy is to scan all network packets (or some representative sample thereof), wherever feasible, to ensure that none contain malware commands or malicious code.

Snort is capable of such packet analysis and is able to compare each packet to a standard set of rules, in addition to customized rules. The WHB implementation uses both the standard Snort rule set, and custom rules written to aid in the detection of real or simulated malicious bot and CC server applications. When a packet triggers an alert, the rule that caused the alert is included in the alert log. The lack of detection of a known threat allows us to determine if the current set of Snort rules will detect the botnet or if new custom rules are required.

Complex rules can be built to trigger alerts on any number of alert triggering conditions. “Snort can perform protocol analysis and content searching/matching. It can be used to detect a variety of attacks and probes, such as buffer overflows, stealth port scans, CGI attacks, SMP probes, OS fingerprinting attempts, and much more. It uses a flexible rules language to describe traffic that it should collect or pass, as well as a detection engine that utilizes a modular plug-in architecture” (SourceFire, 2010).

SLINGbot and Enhanced SLINGbot

SLINGbot (A System for Live Investigation of Next Generation Botnets) and Enhanced SLINGbot are tools engineered in the Python programming language on the Linux operating system. They were provided by Raytheon’s BBN technologies and Mitre Corporation, respectively. The research done by BBN for SLINGbot is summarized by Jackson et al. (2009). Since these tools were built with Python on a Linux system, these same systems are likewise used in my research.

The benign malicious botnets I use in my research are those provided by Mitre Corporation as the sample bot networks built upon three separate bot code bases (Mitre Corporation, 2010). All three are implementations from the Enhanced SLINGbot framework, and all three communicate with HTTP messages. The first is the Sample Botnet, the second is the BlackEnergy Botnet and the third is the Torpig Botnet.

All three adversarial botnets described here are used in each of the experiments in order to provide a variety of different networks for detection purposes. The Sample botnet provides a very simple plain-text message base that proves trivial to detect. The BlackEnergy botnet is valuable in that the sent commands are encrypted, thus increasing the difficulty of detection. The Torpig implementation adds the challenge of detecting multiple servers within a given implementation of this network.

It is expected that detection with the WHB will work for any botnet implementation discussed in chapter II, whether it is HTTP, DNS, IRC, or P2P, since all varieties have one thing in common, they all send and receive network messages. Since the Enhanced SLINGbot framework is built solely upon an HTTP based communications protocol, as is the WHB, this protocol is the focus of this research effort. It is assumed that the methodology used will apply equally well to any of the other botnet communications platforms.

Sample Botnet

The HTTP_Bot implementation of the Sample Bot is a simple example bot that uses the Twisted Web Python library to build a server and client that communicate through HTTP messages with *post* and *get* calls to the server, with simple reply messages sent back to the bot (Twisted Matrix Labs, 2012). This implementation emulates typical web traffic.

The server is the CC channel and is capable of supporting an arbitrary number of bot clients. The bot clients may be dispersed, as desired, among any number of machines, located on any network connected to the internet. The bots connect to their server using the IP address and port number that the server is listening on. The communication interface is handled by the Twisted library and the Enhanced SLINGbot framework. The Sample Bot and Sample CC server send all their messages in clear text and have only one simple command.

Upon execution of the bot, it connects to the server and sends it a "hello" request. Upon receipt of the "hello" request, the server replies back to the bot with a "Hello there!" response message. The bot continues to send out "hello" requests to the server on a timed loop to simulate bot traffic on the network. This interface allows the bot to send messages to the server in the following form: `response = client.get(command_url)`. When the bot needs to send data to the server the interface, the format is: `response = client.post(command_url, {data_dictionary})`. The `command_url` specifies the function that will be executed by the server and is a unique URL for each function in the server that processes a command. The response will be assigned by the server and may contain any data that the server needs to provide the client to process the command.

Essentially, a bot client's `post` or `get` request to the appropriate URL starts the corresponding command function specified in the server, and the value assigned to the reply is the return value that the server function passes as a remote procedure call return to the bot client. For example, the Sample Bot server, when executed, provides an interface for the bot by creating its hello URL with the command: cc.get("/", hello). This simply provides the HTTP_Bot class with communication interface of the root URL, "/" and with the command hello as defined within the server's file. The bot, upon execution, starts by sending a hello message to the server with the following command: resp = client.get("/"). The command, client.get("/") invokes the hello function in the server which logs the request, then returns the string "hello there!" back to the bot. The return value, "hello there!" is then assigned to the resp variable in the bot code. The bot logs the response from the server and returns ('hello', 10) to the HTTP_Bot framework instantiation, which simply sets a 10 second delay and the next command, hello in this case, for the bot to execute. The sequence diagram in Figure 1 depicts the interaction between the sample bot and CC server.

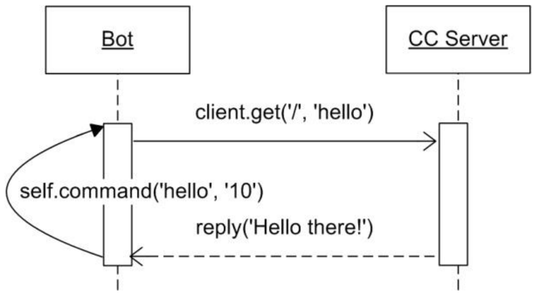


Figure 1: Sample Bot Sequence Diagram

BlackEnergy Botnet

A more capable and robust bot, also built on top of the HTTP_Bot class of the Enhanced SLINGbot framework, is the BlackEnergy botnet was provided by Mitre. The basic implementation does some more interesting things than the sample bot described previously. All of the commands are encoded with a 64-bit encoding from Python's Base64 library before being sent over the network (Python, 2012a).

Before the commands are encoded with Base64, they are first embedded into a BlackEnergy command string, with each part of the string delimited by a hash tag. The string has the following format: header#command#delay#bot_id, where the header is an arbitrary string that is not explicitly used by the sample BlackEnergy bot or CC server. The command section contains the name of the next command the bot is to execute, delay is an integer representation of the number of seconds the bot should wait before executing the command, and the bot_id is a unique ID number provided by the bot when it connects to the server, and when it passes a reply to the server with the stat command.

This command string is then 64-bit encoded and sent to the bot as the return value of the handle_stat function from the server. After the encoded URL is posted, the bot reads the posted message, decodes and executes the command, and returns a relevant reply back to the server in clear text. A simple upgrade would allow the bot to also encrypt its reply messages to the server.

This BlackEnergy bot network simulates real-world bot activity by generating randomly chosen commands, at random intervals, to do the following: send a stat message to the bot, that responds with the bot's unique ID number; and send a file to the bot containing a random set of text characters, compressed by Python's Zlib, from the server to one of the bots in the network (Python, 2012b).

The random file generation capability is representative of the fact that a botmaster may choose to send any file of his or her choosing at any given time. The fact that the server randomizes when to send is also typical of a botnet, since some bots and their servers may lie dormant for great lengths of time, with sporadic periods of activity at predetermined or otherwise intermittent times.

The CC server sets the stat function interface by calling cc.post('/dot/stat.php', handle_stat). This allows the bot to post its reply to the web server by calling client.post('/dot/stat.php',{'id':BOT_ID}). This command posts the bot's ID number to the server in the form of a Python dictionary object. The Python's dictionary object may contain an arbitrary number of entries as key-value pairs (in this case the key is the string 'id' and the value is the BOT_ID number randomly generated by the bot at runtime), and each entry may contain any data type, such as a file, string, list, etc., which provides the ability for the bot to post any data required by the server. The dictionary entries are passed to the server who can then read the necessary entries to complete the requested command.

The resp variable is read by the bot and contains the server's reply comprised of the encoded command argument from the server. The bot uses the Python Base64 library to decode the command argument, then parses the argument and returns the next command and delay provided by the server back to its parent class, the HTTP_Bot instance, which processes the next command.

The actual BlackEnergy botnet variants found in the wild are "…an HTTP-based botnet used primarily for DDoS attacks by the Russian hacker underground" (Koo et al., 2011). Koo further explains that the BlackEnergy bots connect only periodically to their CC servers in order to query the server for the next command to execute. They do not maintain a persistent connection with the server, but rather connect to send information or queries and immediately disconnect once the information is sent. This periodic connection is also typical of the HTTP protocol used by standard web servers and by the other HTTB botnet variants.

Torpig Botnet

The Torpig botnet, also built on the HTTP_Bot framework, consists of a bot, an Inject server, a CC server, and a Mebroot server. Each of the servers may be implemented on separate machines, and they each provide different services to the bot. This layout provides resiliency by allowing a distributed server base, as well as a distributed botnet. The drawback to this topology, however, is that if any part of the server is unavailable, the bot network will either be only partially functional or not functional at all.

In a real-world implementation of the Torpig botnet, each server has a specific purpose (Kemmerer, 2012). Target pages are chosen, such as bank login pages, and when a target is visited, the Torpig CC server issues a request to the inject server, and ports its version of the web page into the victim's browser. As you can guess, the login credentials are returned to the Torpig CC server and subsequently used to defraud the victim. The Mebroot CC server installs the Mebroot rootkit on the victim's machine usually via a Drive-by-download attack. If successful, this allows the botmaster to take full control of the victim's machine.

The simple implementation provided with the Enhanced SLINGbot framework emulates some of the traffic created by the real world botnet. The bot will receive simulated code updates from the Mebroot server. The bot receives a fake web form from the Inject server, and it sends phony stolen log in credentials to the Torpig CC server.

Before the bot or servers send any messages, the commands or data are encrypted by the Torpig Encryption Algorithm. This algorithm first applies a keyed exclusive or (xor) sequence, and then encodes the message with Python's base64 library. All three

Torpig servers and the bots share the same encryption key, and are thus able to decrypt the messages by first decoding the message with base64, then running the decoded message through the same xor sequence. This encryption scheme presents challenges to the detection of this botnet's messages. Fortunately, each unique unencrypted message directly maps to a single unique encrypted message, and similarities in the messages can be keyed in on with detection rules. Figure 2 shows this algorithm in detail.

```
        Enhanced SLINGbot Torpig Encryption Algorithm
TORPIG_ENCRYPTION_KEY → "1234ABC"
xor(string)
    data → ""
    for character in string
        for key_char in TORPIG_ENCRYPTION_KEY
          character → binary.xor(character, key_char)
        data → data + character
    return data

def encrypt(input_string)
    return_string → xor(input_string)
    return Python.base64.encode(return_string)

def decrypt(input_string)
    return_string → Python.base64.decode(input_string)
    return xor(return_string)
```

Figure 2: Enhanced SLINGbot Encryption Algorithm

The White-hat Bot

The WHB is a distributed network monitoring tool that uses the Snort IDS to detect botnet threats. Any alerts triggered on the networked nodes are returned to the CC node. At the CC node, these alerts are analyzed to build a complete picture of any malicious botnet threats found within the network. From the CC node, new detection rules can be created and distributed to detect novel threats. Some of the pertinent functionality of the WHB is described next.

The WHB is implemented by building upon the BlackEnergy Bot example application provided in the Enhanced SLINGbot framework from Mitre. Significant modification and the addition of substantial new functionality have resulted in the WHB in its current form. Included in this added functionality is the ability to send and retrieve files between the bots and server, the ability to traverse bot host file systems, to execute arbitrary code on bot client machines, to start the Snort IDS, to update Snort with custom rules used to detect novel botnets, to retrieve a list of all executing processes and network packet capture logs, to correlate the logs into a map of the infections found within the network, etc. The primary goal of adding this functionality is to enable the WHB with the capabilities necessary to detect and capture bot binaries. This is for the purpose of the creation of new botnet detection rules and strategies in order to combat novel bot variants on a user's machine, server machines, or on honeypot machines or networks.

Highlights of some of the relevant bot functionality of the WHB created during this research effort follows.

Log files

Each bot and CC server creates a log file that contains a record of each command sent/received and the actions taken as a result. The limited log functionality as implemented by the authors of Enhanced SLINGbot only printed log messages to the terminal. Functionality was added to enable this data to also write to a text file output in addition to the terminal. Also, a more verbose logging scheme is implemented to more fully document all of the actions of each bot and server instantiation.

Set bot's ID number from the CC server

The server is capable of setting and resetting each bot's individual identifier as desired. This was accomplished by creating a setbid function call that the server can invoke upon the bot with the argument of the new bot ID. Initial trial runs with 90+ nodes have proven this option to be useful, by allowing the botmaster to rename nodes so that they have useful identities that relate to the physical machine on which they operate.

Return system info

A sysinfo command returns the following information about the system the bot instance runs on: computer name, OS type, release number, version, and machine architecture. A sample return looks like this:

```
                              sysinfo
******************************************************************
2012-03-15 10:09AM - System information
OS type: Linux
Computer name: gubuntu
Release: 3.0.0-16-generic
Version information: #28-Ubuntu SMP Fri Jan 27 17:50:54 UTC 2012
Machine architecture: i686
******************************************************************
```

Figure 3: Sysinfo Command Return

All command modifier

The botmaster uses the all modifier before a command to specify that command to be sent to each connected bot. This simulates a multicast message to all live bots in the network, thus reducing the workload for the human operator when all bots are to complete the same task. This is implemented by keeping an up-to-date list of online bots in the server application, and when a message is intended for all bots, the server loops through the list, sending the same command to each bot individually by implementation of an at-most-once methodology.

Multicast scalability

Analogous to sending a single command to all bots, sending a single command to a specific single bot can be achieved by appending the bot ID before the command, or by typing the command at the interface when that specific bot is currently at the head of the FIFO command queue. If it is not the case that the intended recipient is at the head of the queue of the CC server, then the server must cycle through each connected bot until it reaches the target bot. These scalability limitations of the WHB have not posed an issue during any of the experiments, and it is known that for a WHB network with 96 nodes that there are no significant performance degradations. The limitations of scalability will become problematic at some arbitrarily large unknown network size. The problem is discussed in chapter V and a remedy is deferred to future work.

Command interface

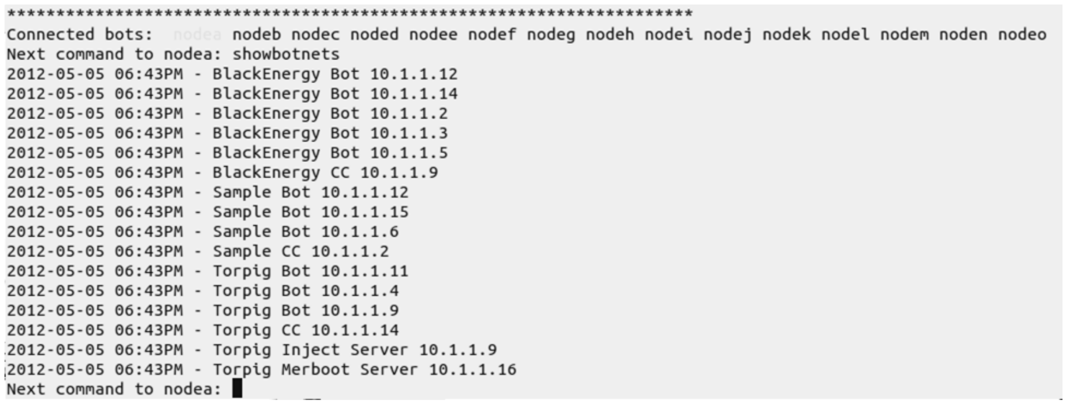
The user-friendly server interface makes it simple to communicate commands to a specified bot, or to propagate a command to all bots. The text in the list of connected bots contains the current bot highlighted in yellow font (gray-scaled in this text):

```
*********************************************
Connected bots: 7078 4768 9272
Next command to 7078:
*********************************************
```

Figure 4: WHB Command Terminal

Botmap and showbotnets commands

The botmap command, sent with the all command modifier, propagates to all the connected WHB bots. The bots capture the current state of their Snort alert logs and return these log files to the CC server. Once all the logs are retrieved, the showbotnets command parses the alert logs and builds a log containing the consolidated data for all the malicious botnet applications found within the network. The list that is printed to the terminal contains the bot application type, and the IP address for each bot or CC server located on each machine. Additionally, a more verbose log file is produced by this command that displays the name of each botnet application type found, the IP addresses of each application, the ports used by each IP address, the number of messages sent from each port, and a sum total of the number of messages sent from each bot instance at each given IP address. An example of the output of the showbotnets command is shown below in Figure 5.



```
**********************************************************************
Connected bots:  nodea  nodeb nodec noded nodee nodef nodeg nodeh nodei nodej nodek nodel nodem noden nodeo
Next command to nodea: showbotnets
2012-05-05 06:43PM - BlackEnergy Bot 10.1.1.12
2012-05-05 06:43PM - BlackEnergy Bot 10.1.1.14
2012-05-05 06:43PM - BlackEnergy Bot 10.1.1.2
2012-05-05 06:43PM - BlackEnergy Bot 10.1.1.3
2012-05-05 06:43PM - BlackEnergy Bot 10.1.1.5
2012-05-05 06:43PM - BlackEnergy CC 10.1.1.9
2012-05-05 06:43PM - Sample Bot 10.1.1.12
2012-05-05 06:43PM - Sample Bot 10.1.1.15
2012-05-05 06:43PM - Sample Bot 10.1.1.6
2012-05-05 06:43PM - Sample CC 10.1.1.2
2012-05-05 06:43PM - Torpig Bot 10.1.1.11
2012-05-05 06:43PM - Torpig Bot 10.1.1.4
2012-05-05 06:43PM - Torpig Bot 10.1.1.9
2012-05-05 06:43PM - Torpig CC 10.1.1.14
2012-05-05 06:43PM - Torpig Inject Server 10.1.1.9
2012-05-05 06:43PM - Torpig Merboot Server 10.1.1.16
Next command to nodea: █
```

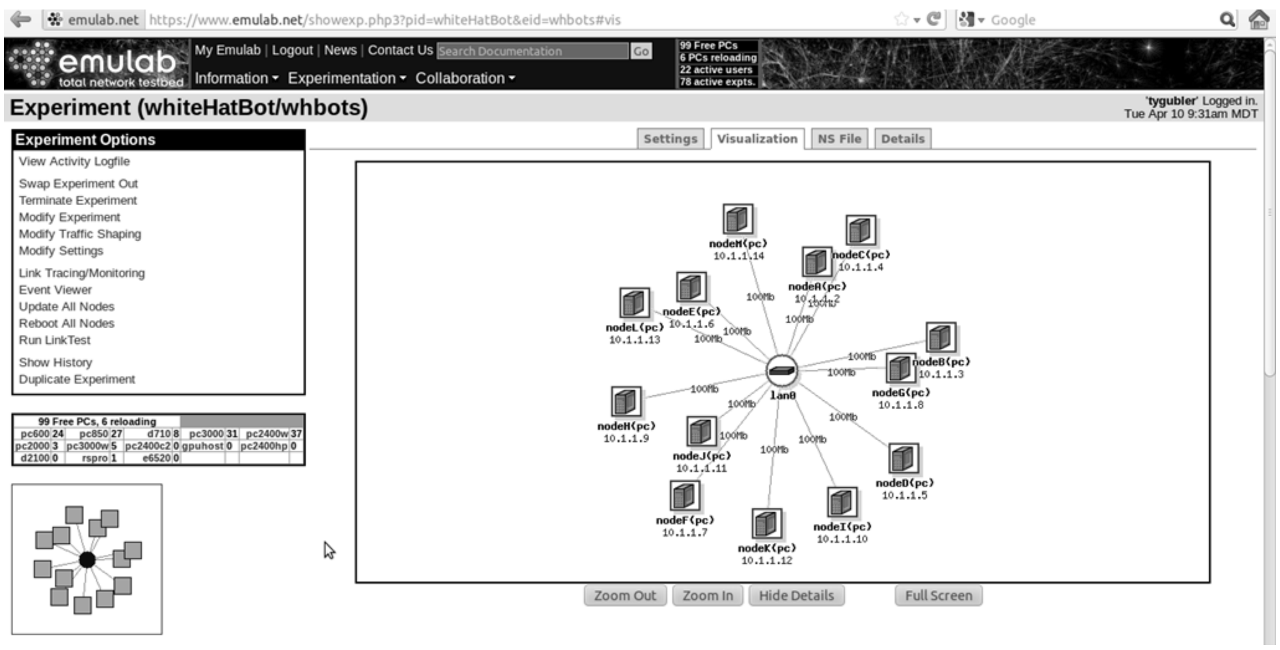
Figure 5: Showbotnets output

Emulab Testbed

Emulab is a network testbed located within the University of Utah (Emulab, 2012). It provides a large number of customizable network nodes that can emulate virtually any network topology specified, within the limits of available nodes. Each node can be customized to the user's needs by preloading any specified OS, along with any other software provided by the user. This allows the user to create a large heterogeneous or homogenous network of nodes, configured with network switches, hubs, and links in whatever topology is desired. When an experiment is started, the specified network is created and each node is booted and loaded with the specified software, allowing for research experimentation in a virtualized environment.

The Emulab website provides a Java applet that allows visual creation of the network using drag-and-drop to add nodes into the network, to create network links, and to specify OS and other optional software to add to each node. Given a network as input, the application generates a Name Script (NS) file that is used to create and run the experiment. The file can then be further modified textually as needed before starting the generation of the emulation of the trial. Further, the application provides an interface to access each node with SSH during the process of an experiment by double-clicking the icon that represents the node of interest.

Figure 6 shows a view of an active 13-node experiment running on the Emulab testbed.



99 Free PCs, 6 reloading									
pc600	24	pc850	27	d710	8	pc3000	31	pc2400w	37
pc2000	3	pc3000w	5	pc2400c2	0	gpuhost	0	pc2400hp	0
d2100	0	rspro	1	e6520	0				

Figure 6: Emulab Experiment Web Interface

From this browser interface you can see a visual representation of the experimental network as it is running. Visual options allow you to modify and restart the current experiment, terminate it, check the active status of each node, access the boot logs of each node, and perform a number of other useful administrative functions. This allows you to monitor the network at a glance and facilitates easy problem identification and remediation.

Launching Botnet experiments on Emulab

Emulab allows creation of a Network Simulation (NS) script used to specify network topology, the operating system to load on each node, and any software that should be pre-loaded into a specified directory location for each node specified. Additionally, a single command can be issued for each node to start an application at node boot time. During experiment creation, Emulab places a text file into the directory

/etc/hosts, on each node, that contains the IP address and node name of each node in the experimental network.

I created a Python script, *updatehosts.py*, which takes as input the Emulab provided hosts file, and the botnet configuration text file. The application starts by reading the Emulab hosts text file, line by line, and builds a list of IP address, node name pairs. Next, the script parses the botnet configuration file, which tells the application which malicious bot applications should be installed on which nodes. Then, the script determines the operating system's node name by calling the OS system call: os.uname()[1] which returns index 1 of a list of system attributes, index 1 being the computer name of the node.

Using this information, *updatehosts.py* uses a series of decision structures to determine which bots and servers are to be started based upon the operating node's name, thus executing the appropriate applications. This also allows bots to connect to the appropriate server by inserting the appropriate IP address of the server node(s) as input arguments to start the bot applications as necessary. If the node is designated as a server, it will immediately start the applicable server process on the specified port. Any bot execution is set to be delayed 120 seconds to provide a reasonable chance for the corresponding server to boot prior to a connection attempt. After the script starts all the applicable malware for a given node, it then starts the WHB client software and Snort application.

This delay strategy has proven to be more than sufficient because Emulab does synchronize the event start script such that no applications begin execution on any node until all nodes have booted and are ready. The delay my script imposes ensures that all the CC servers have ample time to boot on every node prior to the bot applications.

In Emulab, it is possible to create a barrier to guarantee a boot ordering to ensure that all servers are up before any bot is allowed to execute. This has not proven necessary, since the current method accomplished the desired result. If problems result in the future from the current configuration at boot time, such a mechanism will be required. With 96 nodes, the current implementation is a viable solution with a 100% success rate for a simple boot test that was conducted during development of the WHB. No tests were performed to measure the failure rate if the network grows to a larger scale. Since a larger network of more than 96 nodes is beyond the reasonable limits of the Emulab testbed, given its typical workload, this problem will not need to be remedied during the course of this research, but rather a solution is deferred to future work if it becomes necessary.

Command and Control (CC) terminal

In order for a human operator to take control of the WHB CC server, the server is manually started after the experiment has been created and all of the other software is automatically started. The 120 second delay executed by the *updatehosts.py* script provides sufficient time for the botmaster to connect via Secure Shell (SSH) into the control node to manually start the WHB CC server application prior to any bot's attempt to connect. This allows the botmaster to have full control of the CC server in an SSH terminal, and has proven successful in all trials.

Experimental Setup and Metrics

All experiments were all conducted on the Emulab testbed. I conducted trials using BotHunter, the WHB with the standard Snort rule sets, the WHB with custom built rules, and the WHB with random bot network configurations. The setup for these trials is explained in this section.

BotHunter Baseline Trial

Figure 7 shows the general network configuration of this baseline trial experiment.

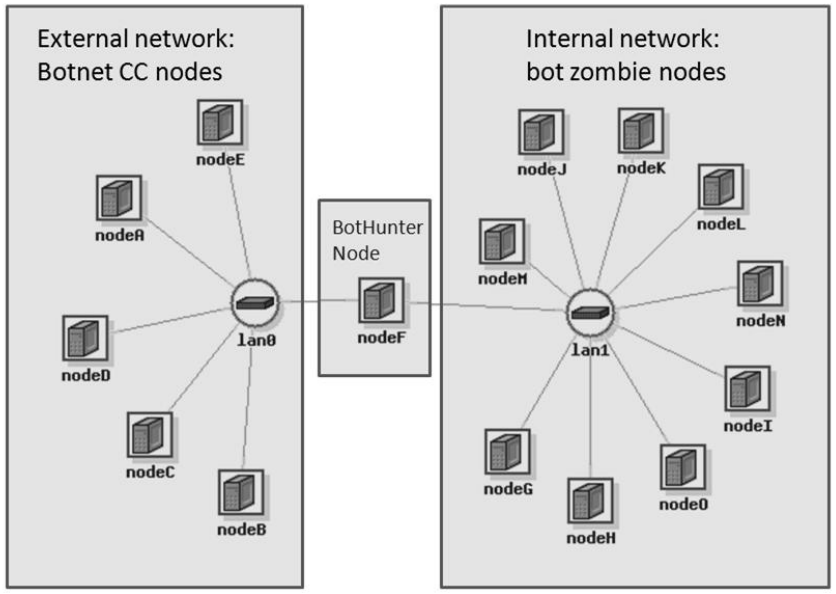


Figure 7: BotHunter Trial Network Diagram

BotHunter is installed on nodeF to scan all the traffic that traverses between the two networks. The trusted network that BotHunter protects consists of the bot infected nodes on the right, connected directly to LAN1. All of the nodes on the left side of the graph, nodes A through E, are running the CC servers for each of the three botnet varieties. LAN0 connects this network to the BotHunter node, nodeF. NodeF is also connected, via a separate network adapter, to Lan1. This configuration requires that all traffic between the two networks traverse nodeF, allowing BotHunter to scan all of the traffic sent between the two networks.

This network uses the standard, network monitor node, BotHunter configuration as recommended by the authors of the software. Three bots of each of the three varieties (Torpig, BlackEnergy and Sample Bot) are represented in this trial. At the commencement of the trial, each of the bots connect to each of their respective CC servers and spend an hour of real time sending messages back and forth between the two networks while nodeF scans and analyzes the traffic for botnet activity.

Initial WHB trial with standard Snort Rules

This WHB trial is conducted to determine if the standard set of Snort rules is sufficient to detect novel botnets, such as those represented by the Enhanced SLINGbot malicious botnets provided by Mitre Corporation. A thirteen node experiment is created with the nodes numbered nodeA through nodeM, with the botnet and Snort software installed as specified below in Table 1.

Table 1: 13-node Preliminary Experiment

Node Name	Malicious Bot Software	WHB Software	Snort
nodeA	Torpig Merboot Svr	WHB	Yes
nodeB	Torpig CC	WHB	Yes
nodeC	Torpig Inject Svr	WHB	Yes
nodeD	Torpig Bot	WHB	Yes
nodeE	Torpig Bot	WHB	Yes
nodeF	BlackEnergy CC	WHB	Yes
nodeG	BlackEnergy Bot	WHB	Yes
nodeH	BlackEnergy Bot	WHB	Yes
nodeI	BlackEnergy Bot	WHB	Yes
nodeJ	Sample Bot CC	WHB	Yes
nodeK	Sample Bot	WHB	Yes
nodeL	Sample Bot	WHB	Yes
nodeM	None	WHB CC	No

As shown in Table 1, this preliminary experiment is set with nodeM as the WHB CC node, and with the WHB client software operating on each of the other nodes. Additionally, malicious bot or server code is deployed onto each of the other 12 nodes such that each of the three malicious botnets is fully represented within the experiment. For the purposes of this trial, as the experiment runs, any suspicious network traffic alerts are captured from nodes A through L and further analyzed at the WHB CC node.

This trial is set up in the simple star network topology shown below in Figure 8.

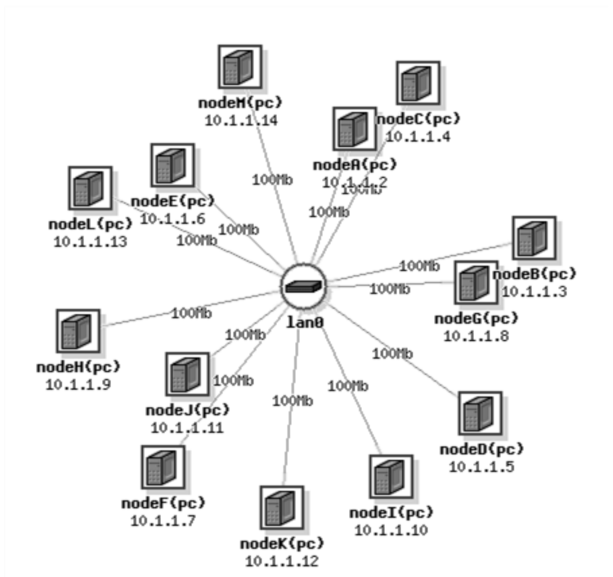


Figure 8: WHB with standard Snort rules trial

WHB trial with Snort custom rules for Enhanced SLINGBot detection

The custom rules trial incorporates custom Snort rules that were created based upon the network traffic generated by the malicious botnets in the prior standard Snort Rules Trial. The network and setup mirrors the setup of the previous trial as shown in Table 1 and Figure 8. These rules attempt to detect each of the malicious botnets in such a way that each botnet may be identified individually. These rules also attempt to create categorized alerts based upon the message type, such as BlackEnergy CC command message, or BlackEnergy Bot stat reply. With the addition of any custom rules that are found to be useful for detection of a specific threat, whenever the WHB is deployed in the future, these rules can be included to attempt detection of any of the malicious botnets listed earlier in Table 1.

Random network experiments

For these trials, I conducted a total of ten experiments using a 15-node star network topology, similar to the 13-node network portrayed in Figure 8. I added the bot networks with random distributions across the network with some tunable parameters.

Within the random setup parameters, I specify the total number of nodes. I also specify the minimum and maximum number of bot nodes of each variety that the script will choose to distribute over the network. For each of the 15-node trials, I set the number of nodes to 15, the minimum number of bots to two, and the maximum number of bots to five. The script is required by default to install one CC node for each of the bot networks, each of which is assigned to a random node. Then, for each variety of bot it chooses a random number of bot nodes, within the maximum and minimum thresholds set, and installs them on arbitrary nodes.

The WHB CC application is installed on the first experiment node, and the WHB clients are installed on every node, including the first node. Then, as described, all the nodes are randomly assigned malicious bot and CC applications. The result is a random malicious network where a given node may contain zero, one, or more malicious bots and/or CC applications. Below, Table 2 displays the output of a 15-node randomized setup with the minimum number of bots set to two and the maximum number set to five for each variety of bot.

Table 2: Random Network Setup

Node	Malware	Malware	Malware	Malware
nodeA	Sample Bot CC			
nodeB	BlackEnergy Bot			
nodeC	Sample Bot			
nodeD				
nodeE				
nodeF				
nodeG	BlackEnergy Bot	Sample Bot	Torpig Bot	Torpig CC
nodeH				
nodeI				
nodeJ	BlackEnergy Bot			
nodeK				
nodeL				
nodeM	BlackEnergy Bot	BlackEnergy CC	Torpig Inject Svr	
nodeN	Torpig Merboot Svr			
nodeO	BlackEnergy Bot	Torpig Bot		

This particular execution of the setup script is illustrative of the random distribution of botnet applications produced by the random network setup script and resulted in a total of five BlackEnergy bots, two Torpig bots, and two Sample Bots.

A randomized setup for each experiment provides a more realistic environment for the WHB to attempt its detection operations. In the previous experiments it was clear up front on which node each malicious bot and CC server operated. In reality we are never going to know in advance where bot software will be placed within our network; hence, the decision was made to randomize the setup.

Assumptions/Limitations

This research assumes that the Snort Intrusion Detection System (Snort IDS) application, including any of its hardware and software dependencies, is pre-installed on each of the analyzed user systems. Also, it is assumed that full permission is granted to do whatever is necessary to the user's system, including full read and write access to any files, applications, binaries, etc that may exist on said machine with no regard to the privacy issues that exist in a real-world scenario in which this type of detection operation would typically apply. Additionally, permission is assumed to install, to modify, and to operate any necessary applications to facilitate detection operations and to capture and analyze any network communication packets that originate from or terminate at said machine.

Since the experimentation involves computing equipment connected to live networks, to include the internet, I am limited to simulated malicious bot networks that are benign in nature and have no method of self replication. This software otherwise simulates network traffic typically generated by software that might be seen in a real-world malicious botnet found in the wild. Using simulated malware is a precaution to ensure there is no accidental release of malicious code to the wild during the process of investigations and experimentation.

Additionally, this simulated malware novel, in that there are no anti-virus definitions, or other SSP applications, that are already preconfigured with signatures created specifically for this software. Any positive detection that results from the existing software trials in these experiments denotes the generality of that software in detecting novel threats.

In this research effort, I concentrate on a network of Ubuntu Linux operating systems, with the bot networks programmed, both the WHB and simulated malicious varieties, in the Python programming language using the Enhanced SLINGbot framework made available to me from Mitre Corporation. Whatever is accomplished on this system is assumed to be representative of what may be accomplished on other systems, using other programming languages, given that all other defensive mechanisms of the systems and those of the networks involved are otherwise similar.

For all the experiments in this study, the nodes each run the Ubuntu Linux version 10.04.1 operating system, with Python version 2.6 and Snort version 9.4.1. Additionally, each node ran zero, one or more of the Enhanced SLINGbot versions of simulated malicious botnets, and the WHB software. BotHunter is used in a preliminary experiment to build a baseline for comparison against other methods of botnet detection. With the exception of the baseline BotHunter experiment, in all of the additional trials, the malware is actively trying its exploits on the systems, while the WHB attempts to detect the presence of the bots by scanning the systems with Snort.

At first, only the standard Snort rules are used to attempt detection. If there are undetected botnets, then a sample of all the traffic is captured and additional rules are created to trigger alerts for the botnet messages. The new rules are deployed and implemented by the WHB. Next, new logs will be gathered based upon any alerts logged by these new custom rules. When the bots return these logs to the CC server, automated analysis of the alert logs is conducted to determine all of the detected botnets is the network and the data is correlated.

Summary

The Snort IDS is an important component of the WHB concept which will aid in the successful detection of novel bot activity. The Enhanced SLINGbot code framework further aids in this research by simulating real-world bots without the risk of the propagation of real-world malicious applications. Utilizing botnet techniques, combined with the abilities of Snort, provides a versatile toolset with functionality for immediate action to detect novel botnets. The Emulab testbed is a useful platform that has successfully facilitated all experimentation to test the scalability and feasibility of the WHB concept. Finally, BotHunter is an existing botnet detection tool that helped to create a baseline of comparison to evaluate the WHB.

IV. Analysis and Results

Chapter Overview

This chapter discusses the results of the experimental trails conducted during the course of this research effort. The first trial discussed is done with BotHunter to set a baseline for the detection abilities afforded by the WHB. The second is designed to test the ability of Snort to detect the novel Enhanced SLINGbot malware variants. The subsequent trials demonstrate the viability of the WHB by applying custom Snort rules as a detection tool.

Results of Experiment Scenarios

BotHunter Baseline Trial

In a baseline trial experiment, the Enhanced SLINGbot malicious botnets were scanned in action by the BotHunter software. The results of this trial are telling, after an hour of botnet operation of all three varieties, BotHunter found none of the traffic to be suspicious and hence triggered no alerts.

Another research study showed BotHunter to be a viable tool that can successfully detect many known botnet variants (Sevy, 2009). In his study, Sevy constructed a network and released the following real-world malicious botnets: Agobot, Phatbot, Rxbot and Sdbot. In addition to these real-world botnets, he created his own novel botnet by modifying and deploying as a botnet a program called Cybercraft. Sevy found that BotHunter easily detected three of the four known real-world botnets. Of these botnets, it was unable to detect his novel botnet creation or the preexisting Sdbot version he tested.

As was Sevy’s experience with his custom novel Cybercraft botnet, I too found that the novel SLINGbot botnets were not detected by BotHunter. This lack of detection revealed a 100% false negative detection rate for the novel botnets I tested.

Initial WHB trial with standard Snort Rules

By running the 13-node trial with all three botnets fully represented, no Snort alerts were triggered by the botnet traffic for the standard Snort rule set for a one hour period. As was the case with BotHunter, the lack of detection showed the result was a 100% false negative detection rate for this trial. Since no alerts were triggered, a representative sample of the network traffic logs was manually analyzed to further refine the detection process. This analysis resulted in the custom rules applied in the subsequent trials.

WHB trial with Snort custom rules for Enhanced SLINGBot detection

This experiment was conducted as a precursor to subsequent testing in an effort to test the custom rules that were developed as a result of the previous Snort standard rule trial. The experiment resulted in successful detection and identification of each type of botnet message created by each bot and CC server, with a few minor exceptions.

When the Black Energy server receives a file download request from a bot, this message contains detectable elements. The packets that contain the actual file returned from the server; however, are not detectable by any type of pattern matching because it is a completely randomized file that is sent as the reply. This file is created with a random generator for each request, thus the entire message is completely different each time. All of the other messages to and from the Black Energy bot are fully detectable, and the bot and server send these detectable messages before and after sending the scrambled file messages. The only thing we have to key in on for the scrambled messages are the IP addresses and ports of the sender and receiver for these messages.

This successful detection will allow for further and more detailed analysis in the random network experiments that follow.

Random network experiments

In all of the 15-node trials, every node running botnet applications was correctly identified as such, with a few minor exceptions. Each experiment was terminated as soon as all detectable bot applications were correctly identified. For this reason, some of the total numbers of alerts may appear small; this is simply because the detected application did not have the time to send many messages prior to termination of the experiment. An overview of each of the 15-node trials appears below in Table 3.

Table 3: 15-node Trial Statistics

Trial #	**Total Bots / Total Servers**	**Elapsed time to full detection in minutes**	**Total Alerts Logged**	**Bots not detected (on their CC node)**	**Bots not detected (multiple on single node)**	**Conservative detection rate**	**Liberal detection rate**
1	9/5	6	106	0	0	100.00%	100.00%
2	12/5	8	177	0	3	82.35%	100.00%
3	14/5	8	164	2	3	73.68%	89.47%
4	13/5	7	139	2	1	83.33%	88.89%
5	11/5	6	127	0	0	100.00%	100.00%
6	12/5	8	188	2	1	82.35%	88.24%
7	13/5	7	157	0	2	88.89%	100.00%
8	12/5	7	146	0	1	94.12%	100.00%
9	12/5	7	138	0	2	88.24%	100.00%
10	11/5	6	104	0	1	93.75%	100.00%

In the two right-hand columns of Table 3, I have delineated two separate detection rates for the bot applications in the trial. The calculation for the conservative detection rate is: $\frac{total\_trial(M_A) - total\_nonDetected(M_A)}{total\_trial(M_A)}$, where M_A are malicious applications used in the trial.

The conservative detection rate is based upon the sum of the non-detection tallies in both non-detection reason categories listed in the table: *multiple bots on single node*, and bots that are undetectable because they operate *on their CC* server node.

To reduce ambiguity and to avoid the double counting of non-detected applications, if there exists multiple bots that also reside on their CC node, they will only be listed under the single non-detection category of *on their CC node***.**

The liberal detection rate only counts a bot as undetected if it resides on its CC node and is determined to be undetected for that reason. The consolidated overall detection rates were 88.17% (conservative), and 96.45% for the liberal rate.

The communications frequencies of the botnets are randomized, with fairly low frequencies. The slowest to communicate, due to their delay settings, are the Torpig botnets. These longer delay settings for initial communications account for the six to eight minute times that the WHB takes to reach full network detection. In a real world scenario, this time could be much longer, or much shorter, depending on the rate of chatter produced by the bot networks under investigation.

The total number of botnet servers remained a constant throughout the trials, three servers for the Torpig botnet, one for the Sample Bot, and one for the BlackEnergy botnet. The number of bots for each variety was randomly chosen between two and five, and the overall number of bots ranged from 9 to 14 for these trials.

With an overall total of 70 minutes of detection time, a total of 1,446 alerts were logged. This resulted in an average of 21 messages flagged during each minute the experiments ran.

The first detection complication occurs because of the HTTP botnet architecture. Given an implementation of this architecture, each of the CC servers always listens on a specified port, making their detection relatively simple. Each bot within a given CC server's network always connects to this CC server's port to send its messages. In contrast, the bot nodes do not maintain a persistent connection with their CC server. Instead, they initiate a new connection each time they send data or new requests to the server. After sending a command, their communications session is terminated. They subsequently reconnect to the CC server again, only after some arbitrary amount of time has elapsed, in order to send their next message. As a result, the bot may be assigned a different send port at the will of the underlying operating system's communications facilities. This phenomenon is especially prevalent with the traffic created by the Torpig bots, and the hypothesis is that this is because they each connect with three different servers over the course of time.

Table 4 contains an example of a node on which only a single Torpig bot operated, in which five different ports were assigned over the period that lasted a total of six minutes, in which time this bot sent a total of 12 messages.

Table 4: Torpig Bot example from 15-node trial 2

Node	Botnet SW	Detected?	Total Alerts	Port	# Msg	Port	# Msg	Port	# Msg	Port	# Msg	Port	# Msg
nodeh	TorBot	Yes	12	42779	2	44583	2	44584	2	45885	4	58604	2

Because of these non-persistent connections, a single bot instance that sends *n* messages to a single CC server may potentially send these messages from *n* different send ports. When the case exists that a single node has two or more instances of a specific bot type operating on it, and each instance shares a common CC server, then it becomes impossible from the alert data to determine if there is only one, or more than one, instance of that specific bot operating on that specific node.

This limitation is only minor since we *are* able to determine that at least one bot of a specific type is operating on a node. If the node is later cleaned of the malware, if only one instance is removed where more instances exist, then additional alerts will be generated when the remaining bot instances re-connect to their CC servers. Also, the malware that is removed is possibly the same software that all instances of the bot on the node use, such that its removal destroys any instances that were there. This is the case with the botnets used in this trial, as configured; however, the application could just as easily be replicated into multiple directories on the node and executed from each of these copies.

Given that we know that one or more bots of a specific type that report to a common CC server based upon alerts triggered is sufficient to justify the use of the liberal detection rate. Though we can't always definitively assign a total number to the bot instances on a node, we can still show that alerts were logged for every instance of each bot of that type that sent out communications packets. For this reason, I favor the liberal detection rate, since multiple bots of a single type who report to a single CC node can arguably be considered identified when only one is detected.

When multiple bots of the same type are on a specific host, we do see an increase in the number of outgoing ports and an increase in the number of packets logged. When these data items are compared to other nodes within the same botnet, this can potentially provide a cue that there may be more than one bot on a node. One can see from the results of the more simple Sample Bot that where there are two bots operating on a given node, there are usually only two outgoing ports used, and where there is only one bot, there is only one outgoing port, and roughly half the traffic. Table 5 shows an example of this phenomenon in action.

Table 5: Sample Bot example from Random 15-node Trial 3

Node	IP	Botnet SW	Detected?	Non-detect reason	Total Alerts	Port	# Msg	Port	# Msg
nodef	10.1.1.7	sampleBot	Yes		6	33368	2	33370	4
nodef	10.1.1.7	sampleBot	No	*2 on node					
nodeo	10.1.16	sampleBot	Yes		4	54396	4		

Looking at Table 5, one might be inclined to claim that both bots on nodeF were detected, and they would be correct in that assumption in this case since we know that by comparing the message alert logs to the random setup log that message alerts were indeed logged for both instances of the bot because there are definitely two bots on the node. Thus it should be safe to state that they were both detected, or should it?

In real life situations we don't have the malware network layout to refer to later to confirm or refute our assumptions based on empirical detection results. The problem with making such assumptions brings me back to Table 4, wherein a single bot instance sent outgoing messages on five different ports. Since this is the inherent nature of HTTP bots, all that can ever really be said about a specific node when alerts are triggered for a specific bot type is that there is at least one bot on that particular node of the bot type identified, but there may also be more. The only way, given the methods used in this research, to definitively determine that more than one bot of a specified type operate on a single node is if the two bots report to different CC servers, and thus belong to different bot networks. Other methods, such as analyzing the running processes on the node can help to determine such a situation.

Another detection limitation of the WHB occurs when a CC server and one or more bots within that server’s domain all operate on the same node. When this occurs, any communications between the bot(s) on that node and CC server do not result in network packages being sent across the wire. Instead, the underlying communications facilities deliver these messages between the applications through the loopback Ethernet adapter. Snort is configured such that it monitors only a single specified network adapter. Snort is capable of monitoring the loopback adapter; however, for the purposes of this research it is monitors the Ethernet adapter that connects the node to the network. No alert is triggered for this reason.

One potential solution to this problem is to execute two instances of the Snort application, one to monitor the external network adapter and the second to monitor the loopback interface. Since each instance of Snort is only capable of monitoring a single adapter, this is the only solution to this problem when using Snort as the payload. This is a resource intensive solution which was not attempted in this research effort. The feasibility of this option was confirmed by reconfiguring Snort to monitor the loopback adapter of a node to subsequently capture the packets generated by a Sample botnet that operated on the same node.

This limitation is also minor in scope because a botnet that operates on a single node is ineffective and will likely not accomplish anything useful for the botmaster. Additionally, we have shown that it is possible to detect this situation by monitoring the traffic on the loopback interface. If a bot and CC server both operate on the same node in a useful botnet, then the CC server or the bot will also necessarily communicate with other nodes, thus producing network packets that can be used for detection purposes.

With these packets we can detect the server, and any of the other nodes with which it directly communicates. If the CC server and all of the non-CC node bots are removed from the network, the bot on the old CC server node will likely be rendered useless, unless it has a backup server with which it will later communicate, in which case it can later be detected when it sends messages to its other server.

For the reasons mentioned, when monitoring network packets for botnet application traffic, neglecting to scan the traffic that may appear on the loopback adapters of monitored nodes does not pose much of a threat. If a bot connects to a CC application on the same node, that bot is of little or no use to the botmaster and is likely inconsequential to the success of the botnet. Neglecting such an application, while undesirable, is unlikely to cause any real harm.

The exception to this problem for the botnets tested is the Torpig bot, which communicates with three different servers. If the bot operates on one of the nodes with which one or two of its servers resides, it is still detectible through its communications with the server(s) located on other node(s). However, like the single-server configuration, if the bot and all three servers operate on the same machine, then that bot instance produces no network packets, but the bot then becomes next to useless to the botnet given no other means of communication with other nodes.

Figure 9 illustrates the results of the first 15-node random setup experiment.

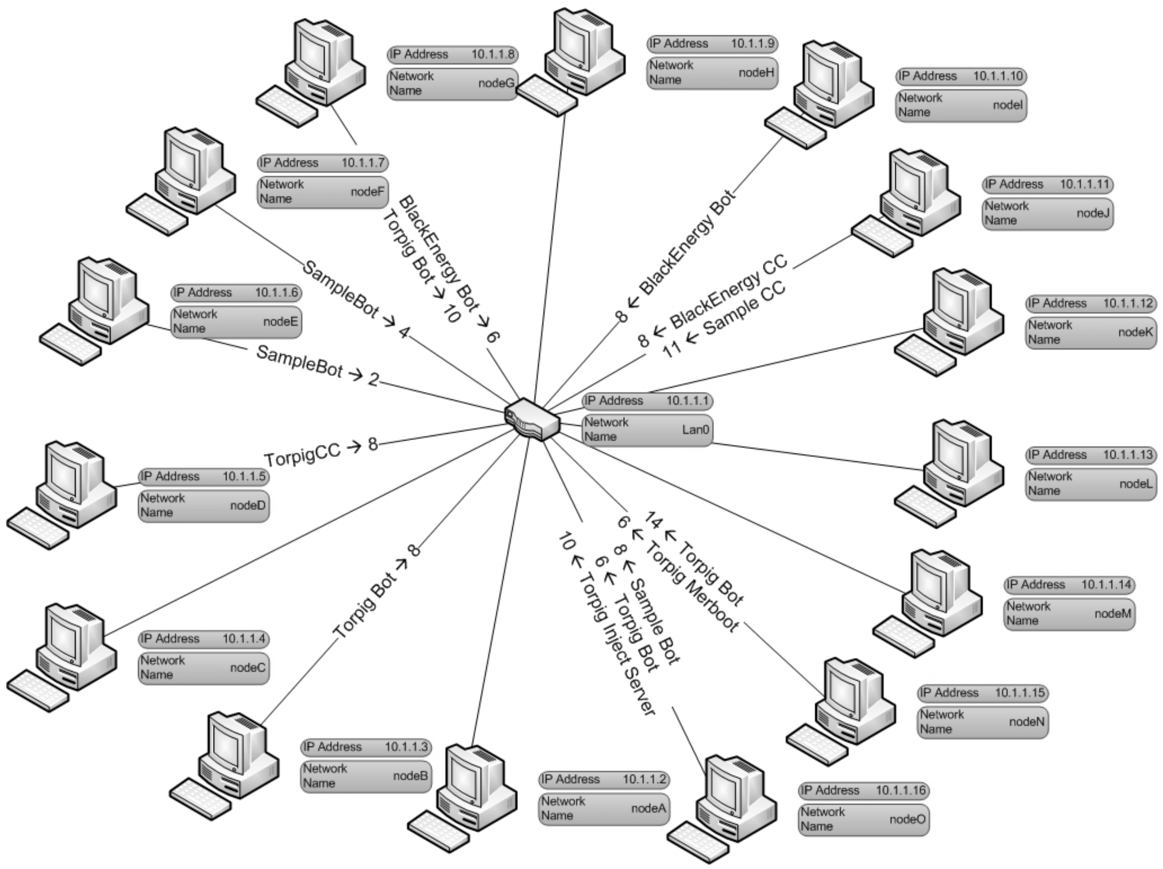


Figure 9: 15-node random botnet trial 1

In this trial, there were a total of 106 Snort alerts triggered, properly identifying all of the bots and CC servers in the network. Two of the three Torpig server nodes also have Torpig bot applications running on their respective nodes, both nodes *N* and *O*. As discussed earlier in this section, these bots were detected because they both communicate with the other two servers through the network, while communicating with the servers on their respective host nodes through local message passing handled by the local Kernel and OS.

The total number of alerts triggered by each of the applications on each of the nodes is noted in the diagram, after the name of the application, and after the arrow that points in the direction of travel for the messages.

Investigative Questions Answered

Will the WHB be more effective than current methods, such as typical bot-detection software, to locate zero-day malicious bot network applications running on infected hosts? If so, what methods and best practices must be applied to make this possible?

This study has shown that the WHB, as it is currently implemented, can root out botnet threats on a distributed system of nodes without regard to the limitations of administrative boundaries. With the application of custom Snort rules, combined with the distributed network of the WHB, the threats that can be detected are nearly limitless. As alert data is returned to the CC node, a complete picture of the threats detected within the network is assembled and presented.

Currently, manual intervention may be needed when novel threats go undetected by the currently applied set of Snort rules. This manual intervention is required to ensure that novel threats are correctly pinpointed such that new rules can be crafted to ensure their future detection without risking false positive alerts being triggered also. Once a new rule is properly crafted and sent out to the WHB network, all instances of that threat will be detected, and alerts will be triggered the very moment the malicious application attempts to communicate with any other node, whether or not the other node lies within, or outside of, the detecting WHB network.

Will the distributive nature of botnets and the methods employed by malicious bot masters be beneficial in the detection of novel bot threats?

This study has revealed the value of the distributed nature of botnets in spanning multiple domains has proven useful in the detection of novel botnets. While the results of the alerts logged on an individual node can be useful in detecting botnets, this study has shown that the combined alerts from a distributed network of nodes can aid in the discovery of the entire malicious network. Further, from the centralized CC node, additional analysis of message traffic can be accomplished to help identify novel threats, and to build detection rules that can then be immediately distributed out to all the WHB nodes.

Can the benefits of current tools be leveraged and amplified by their use in a distributed bot network? If so, how?

This study has shown the viability of using distributed botnet technology as a means of detecting and identifying malicious botnets. In this study, the WHB was used with the Snort IDS; however, it can easily be reconfigured to use any tools that will aid in the efficiency with which it is able to detect novel bot threats. This flexibility adds to the usefulness of the WHB, since tools are updated and new tools are created constantly.

Summary

The final WHB trials conducted in this study produced a liberal detection rate of 96.45%. This is a significant finding that warrants further investigation into and refinement of the WHB concept as a distributed bot network detection tool. Baseline comparison trials using BotHunter and Snort with its standard rule sets resulted in no successful detections of the novel threats studied. This lack of detection is a call for better methods. The successful detections resulting from the WHB implementation with custom rules show that this is a promising endeavor that should continue to be pursued.

V. Conclusions and Recommendations

Chapter Overview

This chapter focuses first on the conclusions of this research effort to include the advantages of the WHB. Then, I discuss recommendations for future research. Finally, I conclude this study.

Conclusions of Research

Advantages of the WHB

One advantage that the WHB offers is that the packet capture and initial analysis of packets is accomplished at each node individually by the Snort application. The typical deployment of Snort in botnet detection requires a single Snort machine to be dedicated to scan all the traffic within a given domain or network. This creates a bottleneck in the network since it requires that all traffic be routed through the Snort node. This may cause serious delays within the network, especially during periods of peak traffic. When the Snort monitor node becomes overwhelmed with network traffic, Snort will begin dropping packets to conserve bandwidth. The underlying OS may also become overwhelmed with traffic, causing dropped packets and potentially an inordinate number of message traffic failures. This may effectively be seen by the network users as a DDoS attack, when in reality it is malware detection software causing the slowdown. In contrast, by installing Snort on individual nodes, it thus increases the workload for each node only slightly by requiring each to scan its own traffic, but it avoids the bottleneck effect of a single monitor node.

Another advantage gained by the WHB distributed employment of Snort is the fact that the CC node can monitor the results of scans that reach a potentially limitless number of administrative domains and networks. Then, the alerts from these distributed nodes can be aggregated and used for further analysis at the CC level as necessary.

A useful feature of Snort that will be leveraged by the WHB in the future allows for the logging of alerts to a standard port, which would allow for each node to report its alerts to the CC node in an automated fashion such that a web interface or any number of other possible alert notification systems can display these alerts in real-time to the CC node. The current configuration only sends alert notifications in bulk and on demand to the CC server, which will cause the network traffic to have large bursts when the CC requests alert updates from a large network at the same time. This real-time forwarding of alerts, as they are triggered and logged, will likely result in a more steady and moderate stream of traffic, without large bursts.

Advanced rule refinement, to find more generalized patterns instead of specific signatures, will aid in the detection of unknown suspicious activity and is an advertised feature of Snort. Non-specific rules are likely to be more prone to false positive alerts and false negative (no alert logged) responses. Since this is the case, these alerts will be logged with a lower priority than known specific threats. These lower priority messages must be further analyzed to determine whether they are a viable threat or not. If they are found to be viable, then new rules can be created to pinpoint the new threat directly, thus placing future messages from that threat into a known flagged category.

The question about false negatives is open ended and requires a priori knowledge of the threat to detect any false negative condition. This is where simulated malware can aid in the process of rule development to overcome false negative conditions. With malware that the researcher creates, the message patterns of this software can be studied and dissected safely. If the malware resembles a future novel real-world exploit, the rules created for its detection may also be effective in the detection of the emerging threat. While this methodology does not provide any guarantees, since it is impossible to anticipate all future threats, it can prove useful if the created rules are general enough to spot threats that are similar in nature.

Recommendations for Future Research

Infection Vectors

One area that was not addressed in this research effort was infection vectors. Potential exists for their application in criminal cases where law enforcement has authorized the invasion of specified IP addresses or networks in order to take down a specific threat. If duly authorized, a directed spreading mechanism could be employed to propagate the WHB to any of the hosts that are authorized in order to covertly analyze said machines and networks for malicious activity in order to find evidence to press charges against the criminals responsible, and ultimately to dismantle the botnets involved. Some of the infection vectors used in real-world malware could provide the WHB with this capability, where authorized by law.

Data Mining

Further research into data mining the aggregated Snort logs in order to detect botnets is warranted. I hypothesize that this is likely one of the best ways in which to successfully move forward with the detection of novel botnet threats. All botnets, regardless of topology, must communicate and hence the best way to detect novel bot threats is through the communications patterns they create. Further research in this area will be able to refine and generate the most capable and robust botnet detection algorithms possible.

Further research in this area may ultimately result in processes that can help to reduce and potentially eliminate the need for manual intervention in order to create new rules to pinpoint and detect novel threats as they arise. When generalized rules trigger alerts, automated processes, if they are refined enough, may be able to create accurate custom rules to precisely categorize a novel threat for future detection and eradication.

My hypothesis is that this is an open-ended problem that may never be fully solvable since anticipating every future novel application that communicates with other applications, in an effort to determine whether it is malicious or not, simply cannot be done. Since the combined knowledge of SSPs has revealed only a finite number of malicious applications, it stands to reason that there also exists some finite number of unknown malicious applications at any given point in time. As time progresses, the number of known and unknown malicious applications changes as threats are discovered and as novel threats are created. As long as the number of unknown threats is greater than zero, then the problem remains unsolved. If the number ever reaches zero, it is unlikely to remain zero for long before a novel malware application is created and deployed that is undetectable given the current set of detection algorithms, signatures, and rules.

Scalability Limitations

The multicast scalability of the WHB is currently limited because of the configuration of the bot and server code. The way communications work is that each bot connects with an initial command, `stat`, in which the bot reports its ID number to the CC server. The CC server then queries the botmaster for the next command to be sent to the bot. As each bot connects with a `stat` command, the CC server will answer each call in the order it arrives. The problem with this configuration is that in order to get a response back from a specific bot, the application must essentially cycle the `stat` requests of each connected bot until the desired recipient bot is reached, then it must again cycle through all *n-1* bots before receiving a response to the message sent. This will lead to unacceptable delays as *n* grows very large; therefore, a more fitting solution is needed in order to deploy the WHB on a very large scale.

A simple solution would be to allow each bot to be individually contacted by the server when necessary, but the problem is that the server never initiates contact with a bot, but rather waits for the bots to contact it creating a FIFO message queue. Direct addressing of a bot without cycling through the queue and sending network messages to each bot in the list in order to locate the target bot would be more scalable. This is really only an issue when targeting a specific bot when there are large numbers of bots connected. An initial trial with 96 WHB nodes indicates that the issue does not cause significant delays at this size, and the responsiveness of the botnet does not suffer significantly at such a network size. Since the issue is not problematic at such a network size, it will be ignored for the purposes of this research effort and deferred to future work.

Summary

As bot and malware defenses advance, so do the tactics of hackers. The need to stay a step ahead is critical. As the interests of terrorist groups such as Al Qaeda, and other adversarial forces in hostile nations continue to gain interest and abilities in botnet technology, the stakes have never been higher to build up and maintain solid defensive strategies to protect our nation's vital assets.

The WHB and other defensive tools can help mitigate the threats that are posed by those who would damage and destroy. Further research and development of network defense is the clarion call of the day.

Bibliography

Bleaken, D. (2010). Botwars: The fight against criminal cyber networks. *Computer Fraud & Security, 2010*(5), 17-19. doi:DOI: 10.1016/S1361-3723(10)70055-5

Chandrashekar, J., Orrin, S., Livadas, C., & Schooler, E. M. (2009). The dark cloud: Understanding and defending against botnets and stealthy malware. Intel Technology Journal, 13(2), 130.

Emulab (2012). Emulab Total Network Testbed. Retrieved 2 May 2012 from http://emulab.net/

Farley, R., & Wang, X. (2010). Roving bugnet: Distributed surveillance threat and mitigation. *Computers & Security, 29*(5), 592-602. doi:DOI: 10.1016/j.cose.2009.12.002

Feily, M., Shahrestani, A., & Ramadass, S. (2009). A survey of botnet and botnet detection. Paper presented at the *Third International Conference on Emerging Security Information, Systems and Technologies, 2009. SECURWARE '09.* 268-273.

Google (2012). AdSense policies: a beginner's guide. Retrieved 8 May 2012 from https://support.google.com/adsense/bin/answer.py?hl=en&answer=23921&topic=1271503&ctx=topic

Grizzard, J. B., Sharma, V., Nunnery, C., Kang, B. B., & Dagon, D. (2007). Peer-to-peer botnets: overview and case study. Paper presented at the *First conference on First Workshop on Hot Topics in Understanding Botnets, 2007*. HotBots'07. USENIX Association, Berkeley, CA.

Gu, G., Porras, P., Yegneswaran, V., Fong, M., & Lee, W. (2007). BotHunter: Detecting Malware Infection Through IDS-driven Dialog Correlation. Paper presented at the *16th USENIX Security Symposium on USENIX Security Symposium*. SS'07. USENIX Association Berkeley, CA.

Honeynet Project (2012). About the Honeynet Project. In *The Honeynet Project*. Retrieved 5 May 2012 from http://www.honeynet.org/about

Hunter, P. (2008). PayPal, FBI and others wage war on botnet armies. can they succeed? *Computer Fraud & Security, 2008*(5), 13-15. doi:DOI: 10.1016/S1361-3723(08)70082-4

Jackson, A. W., Lapsley, D., Jones, C., Zatko, M., Golubitsky, C., & Strayer, W. T. (2009). SLINGbot: A system for live investigation of next generation botnets. Paper presented at the *Conference for Homeland Security, 2009. CATCH '09. Cybersecurity Applications & Technology,* 313-318.

Kaspersky Lab (2012). Heuristic Analysis in Kaspersky Anti-Virus 2012. *Kaspersky Anti-Virus*. Retrieved 30 April 2012 from http://support.kaspersky.com/kav2012/tech?qid=208284682.

Kemmerer, R.A. (2012). How to Steal a Botnet and What Can Happen When You Do. *Department of Computer Science, University of California, Santa Barbara*. Retrieved 15 Mar 2012 from http://www.cs.ucsb.edu/~kemm/courses/cs177/torpig.pdf

Koo, T., Chang, H., & Wei, Q. (2011). Construction P2P firewall HTTP-botnet defense mechanism. Paper presented at the *2011 IEEE International Conference on Computer Science and Automation Engineering (CSAE), 1* 33-39.

Mitre Corporation (2010). Enhanced SLINGBot Emulator v1.0 documentation [Computer software]. Provided electronically by Mitre Corporation.

Muhaya, F. B., Khan, M. K., & Xiang, Y. (2011). Polymorphic malware detection using hierarchical hidden markov model. Paper presented at the *2011 IEEE Ninth International Conference on Dependable, Autonomic and Secure Computing (DASC),* 151-155

Obama, B. H. (2009). Remarks by the President on Securing Our Nation's Cyber Infrastructure. The White House Office of the Press Secretary. Retrieved 10 May 2012 from http://www.whitehouse.gov/the_press_office/Remarks-by-the-President-on-Securing-Our-Nations-Cyber-Infrastructure/

Ollmann, G. (2008). Hacking as a service. *Computer Fraud & Security, 2008*(12), 12-15. doi:DOI: 10.1016/S1361-3723(08)70177-5

Python (2012a). *Python v2.7.2: The Python Standard Library: 18.12.base64*. Retrieved 15 March 2012 from http://docs.python.org/library/base64.html

Python (2012b). *Python v2.7.2: The Python Standard Library: 12.1.zlib*. Retrieved 15 March 2012 from http://docs.python.org/library/zlib.html?highlight=zlib#zlib.

Schiller, C. A., Binkley, J., Harley, D., Evron, G., Bradley, T., Willems, C., & Cross, M. (2007). *Botnets: The killer web app*. Canada: Syngress Publishing, Inc.

Sevy, B. D. (2009). *Using Covert Means to Establish Cybercraft Command and Control*. Master's thesis. Air Force Institute of Technology.

Singh, K., Srivastava, A., Giffin, J., & Lee, W. (2008). Evaluating email's feasibility for botnet command and control. Paper presented at the *IEEE International Conference* *on Dependable Systems and Networks with FTCS and DCC, 2008. DSN 2008,* 376-385

SourceFire (2010). *About Snort*. Retrieved 25 April 2012 from http://snort.org/snort

SRI International (2012). BotHunter: A Network-based Botnet Diagnosis System. *About BotHunter*. Retrieved 19 April 2012 from http://www.bothunter.net/about.html

Still, M., & McCreath, E. C. (2011). DDoS protections for SMTP servers. *International Journal of Computer Science and Security (IJCSS), 4*(6), 537-550

TeamDev (2012). *JxCapture*. Retrieved 30 April 2012 from http://www.teamdev.com/jxcapture/

The Lipman Reports Editors (2010). Cyberterrorism: The invisible threat stealth cyber predators in a climate of escalating risk. Foreign Affairs, 89(6), 24A-24A,24B,24C,24D. http://search.proquest.com/docview/763492003?accountid=26185

Twisted Matrix Labs (2012). Retrieved 15 March 2012 from http://twistedmartix.com/trac.

Wood, P. (2010). Bot wars: The spammers strike back. *Network Security, 2010*(4), 5-8. doi:DOI: 10.1016/S1353-4858(10)70044-1

Wuu, L. & Chen, S. (2003). Building intrusion pattern miner for snort network intrusion detection system. Paper presented at the *IEEE 37th Annual 2003 International Carnahan Conference on Security Technology, 2003 Proceedings.,* 477-484.

Zhu, Z., Lu, G., Chen, Y., Fu, Z. J., Roberts, P., & Han, K. (2008). Botnet research survey. Paper presented at the *32nd Annual IEEE International Computer Software and Applications, 2008. COMPSAC '08.,* 967-972.

Vita

TSgt Tyrone C. Gubler was born 6 February 1975 in Burbank, California, the son of Terry Clayton Gubler and Susan Kay Lindholm. He is married and has 3 children. He joined the USAF as an Aerospace Ground Equipment Mechanic in 1996 and served in that capacity at Mountain Home AFB, ID and Incirlik AB, Turkey until 2000, when he cross trained into the Aviation Resource Management (ARMS) career field. He has served in the ARMS career field at Shaw AFB, SC and Nellis AFB, NV. He completed his B.S. in Computer Information Systems in October 2010 from Saint Leo University, after which he was accepted into the Computer Science Master’s program at AFIT. Following his graduation from AFIT, he will serve as a functional manager for the ARMS program at Gunter AFB, AL where he will employ the skills acquired throughout his career and in his degree programs as he oversees future development and maintenance of the ARMS system.

REPORT DOCUMENTATION PAGE			*Form Approved* *OMB No. 074-0188*
The public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of the collection of information, including suggestions for reducing this burden to Department of Defense, Washington Headquarters Services, Directorate for Information Operations and Reports (0704-0188), 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to an penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number. **PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.**			
1. REPORT DATE *(DD-MM-YYYY)* 14-06-2012	**2. REPORT TYPE** Master's Thesis		**3. DATES COVERED** *(From – To)* September 2010 – June 2012
TITLE AND SUBTITLE The White-hat Bot: A Novel Botnet Defense Strategy			**5a. CONTRACT NUMBER**
			5b. GRANT NUMBER
			5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S) Gubler, Tyrone C., Technical Sergeant, USAF			**5d. PROJECT NUMBER** N/A
			5e. TASK NUMBER
			5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAMES(S) AND ADDRESS(S) Air Force Institute of Technology Graduate School of Engineering and Management (AFIT/ENG) 2950 Hobson Way, Building 640 WPAFB OH 45433-8865			**8. PERFORMING ORGANIZATION REPORT NUMBER** AIFT/GCS/ENG/12-05
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) Intentionally left blank			**10. SPONSOR/MONITOR'S ACRONYM(S)**
			11. SPONSOR/MONITOR'S REPORT NUMBER(S)
12. DISTRIBUTION/AVAILABILITY STATEMENT APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.			
13. SUPPLEMENTARY NOTES This material is declared a work of the U.S. Government and is not subject to copyright protection in the United States.			
14. ABSTRACT Botnets are a threat to computer systems and users around the world. Botmasters can range from annoying spam email propagators to nefarious criminals. These criminals attempt to take down networks or web servers through distributed denial-of-service attacks, to steal corporate secrets, or to launder money from individuals or corporations. As the number and severity of successful botnet attacks rise, computer security experts need to develop better early-detection and removal techniques to protect computer networks and individual computer users from these very real threats. I will define botnets and describe some of their common purposes and current uses. Next, I will reveal some of the techniques currently used by software security professionals to combat this problem. Finally I will provide a novel defensive strategy, the White-hat Bot (WHB), with documented experiments and results that may prove useful in the defense against botnets in the future.			
15. SUBJECT TERMS botnets, malware, botnet detection			

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON** Hemmes, Jeffrey M., Lt Col, USAF ADVISOR
a. REPORT	**b. ABSTRACT**	**c. THIS PAGE**			**19b. TELEPHONE NUMBER** *(Include area code)*
U	U	U	UU	78	(937) 255-6565 jeffrey.hemmes@afit.edu

Standard Form 298 (Rev. 8-98)
Prescribed by ANSI Std. Z39-18